Apéro-réflexion

J.-F. Monin

Univ. Joseph Fourier and VERIMAG
Grenoble, France

JFLA, jan. 2007

Rappels sur Coq

Coq

- logique (formules, règles d'inférence)
- calcul fonctionnel (langage applicatif)
- système de types riche

Démo en ligne

Rappels sur Coq

Coq

- logique (formules, règles d'inférence)
- calcul fonctionnel (langage applicatif)
- système de types riche

Démo en ligne

Rappels sur Coq

Coq

- logique (formules, règles d'inférence)
- calcul fonctionnel (langage applicatif)
- système de types riche

Démo en ligne

Réflexion ou internalisation

Une des idées clé introduites par Gödel (théorème d'incomplétude)

Technique de preuve à la mode en Coq

Réflexion ou internalisation

Une des idées clé introduites par Gödel (théorème d'incomplétude)

Technique de preuve à la mode en Coq

Réflexion ou internalisation

Une des idées clé introduites par Gödel (théorème d'incomplétude)

Technique de preuve à la mode en Coq

Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive *code_of_a_prop*: *Set*:= ...
Definition φ : *code_of_a_prop* → *Prop*
Definition *compute* : *code_of_a_prop* → *bool*

Étape 2: démontrer un théorème de correction
Theorem correctness:
$\forall x : code_of_a_prop, compute\ x = true \rightarrow \varphi x$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !

Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive *code_of_a_prop*: *Set*:= ...
Definition φ : *code_of_a_prop* $\rightarrow$ *Prop*
Definition *compute* : *code_of_a_prop* $\rightarrow$ *bool*

Étape 2: démontrer un théorème de correction
Theorem correctness:
$\forall x : code_of_a_prop, compute\ x = true \rightarrow \varphi x$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !



Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive *code_of_a_prop*: *Set*:= ...
Definition φ : *code_of_a_prop* $\rightarrow$ *Prop*
Definition *compute* : *code_of_a_prop* $\rightarrow$ *bool*

Étape 2: démontrer un théorème de correction
Theorem correctness:
$\forall x : code_of_a_prop, compute\ x = true \rightarrow \varphi x$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !

Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive *code_of_a_prop*: *Set*:= ...
Definition φ : *code_of_a_prop* $\rightarrow$ *Prop*
Definition *compute* : *code_of_a_prop* $\rightarrow$ *bool*

Étape 2: démontrer un théorème de correction
Theorem correctness:

$$\forall x : code_of_a_prop, compute\ x = true \rightarrow \varphi x$$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !



Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive $code_of_a_prop$: $Set := \ldots$
Definition $\varphi : code_of_a_prop \to Prop$
Definition $compute\ : code_of_a_prop \to bool$

Étape 2: démontrer un théorème de correction
Theorem correctness:

$$\forall x : code_of_a_prop, compute\ x \ = \ true \ \to \ \varphi x$$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !



Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive $code_of_a_prop$: Set:= ...
Definition φ : $code_of_a_prop \rightarrow Prop$
Definition $compute$: $code_of_a_prop \rightarrow bool$

Étape 2: démontrer un théorème de correction
Theorem correctness:

$$\forall x : code_of_a_prop, compute\ x = true \rightarrow \varphi x$$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !



Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive *code_of_a_prop*: *Set*:= ...
Definition φ : *code_of_a_prop* $\rightarrow$ *Prop*
Definition *compute* : *code_of_a_prop* $\rightarrow$ *bool*

Étape 2: démontrer un théorème de correction
Theorem correctness:

$$\forall x : code_of_a_prop, compute\ x = true \rightarrow \varphi x$$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !

Réflexion mode d'emploi

Étape 1: internaliser
representer des objets du niveau logique (propositions, preuves) par des données

Inductive $code_of_a_prop$: Set:= ...
Definition φ : $code_of_a_prop \rightarrow Prop$
Definition $compute$: $code_of_a_prop \rightarrow bool$

Étape 2: démontrer un théorème de correction
Theorem correctness:

$$\forall x : code_of_a_prop, compute\ x = true \rightarrow \varphi x$$

Étape 3: jouer
Convertir le but, appliquer le théorème, c'est fini !



Exemples simplistes (1)

Theorem correctness_nat: $\forall n, \mathit{fleches}(S\ n)$
Par induction sur n.

But: $A \rightarrow A \rightarrow A \rightarrow A$
change (*fleches* 3).
apply correctness_nat. Qed.

NB: *compute* est inutile

Exemples simplistes (1)

Theorem correctness_nat: $\forall n, fleches(S\ n)$
Par induction sur n.

But: $A \rightarrow A \rightarrow A \rightarrow A$
change ($fleches$ 3).
apply correctness_nat. Qed.

NB: *compute* est inutile

Exemples simplistes (1)

Theorem correctness_nat: $\forall n, \mathit{fleches}(S\ n)$
Par induction sur n.

But: $A \rightarrow A \rightarrow A \rightarrow A$
change ($\mathit{fleches}$ 3).
apply correctness_nat. Qed.

NB: *compute* est inutile

Exemples simplistes (1)

Theorem correctness_nat: $\forall n, \mathit{fleches}(S\ n)$
Par induction sur n.

But: $A \rightarrow A \rightarrow A \rightarrow A$
change ($\mathit{fleches}$ 3).
apply correctness_nat. Qed.

NB: *compute* est inutile

Exemples simplistes (1)

Theorem correctness_nat: $\forall n, \mathit{fleches}(S\ n)$
Par induction sur n.

But: $A \rightarrow A \rightarrow A \rightarrow A$
change ($\mathit{fleches}$ 3).
apply correctness_nat. Qed.

NB: *compute* est inutile

Exemples simplistes (1)

Theorem correctness_nat: $\forall n, fleches(S\ n)$
Par induction sur n.

But: $A \rightarrow A \rightarrow A \rightarrow A$
change ($fleches$ 3).
apply correctness_nat. Qed.

NB: *compute* est inutile

Exemples simplistes (2)

Similaire, en remplaçant n nat par un couple (l, y)

Inductive ab: Set := a : ab | b : ab.

On définit φ_{ab} par $\varphi_{ab}\,\mathrm{a} = A$ et $\varphi_{ab}\,\mathrm{b} = B$,
et φ tel que $\varphi([x_1; \ldots x_n], y) = \varphi_{ab}\,x_1 \rightarrow \ldots \varphi_{ab}\,x_n \rightarrow \varphi_{ab}\,y$.

Definition compute (l, y) := member y l

Theorem correctness_list:
$\forall l, y, compute\ (l, y) = true \rightarrow \varphi(l, y)$
(par induction sur l)

Goal: $A \rightarrow B \rightarrow A \rightarrow A \rightarrow B \rightarrow A \rightarrow B$
change (φ_{ab} ([a; b; a; a; b; a], b)). apply correctness_list.

(New goal: `compute` ([a; b; a; a; b; a], b) = *true*.)

reflexivity. Qed.



Exemples simplistes (2)

Similaire, en remplaçant n nat par un couple (l, y)

Inductive ab: Set := a : ab | b : ab.

On définit φ_{ab} par $\varphi_{ab}\,\mathsf{a} = A$ et $\varphi_{ab}\,\mathsf{b} = B$,
et φ tel que $\varphi([x_1; \ldots x_n], y) = \varphi_{ab}\, x_1 \to \ldots \varphi_{ab}\, x_n \to \varphi_{ab}\, y$.

Definition compute (l, y) := member y l

Theorem correctness_list:
$\forall l, y, compute\ (l, y)\ =\ true\ \to\ \varphi(l, y)$
(par induction sur l)

Goal: $A \to B \to A \to A \to B \to A \to B$
change (φ_{ab} ([a; b; a; a; b; a], b)). apply correctness_list.

(New goal: `compute` ([a; b; a; a; b; a], b) = *true*.)

reflexivity. Qed.



Exemples simplistes (2)

Similaire, en remplaçant n nat par un couple (l, y)

Inductive ab: Set := a : ab | b : ab.

On définit φ_{ab} par $\varphi_{ab}\,\mathrm{a} = A$ et $\varphi_{ab}\,\mathrm{b} = B$,
et φ tel que $\varphi([x_1; \ldots x_n], y) = \varphi_{ab}\,x_1 \rightarrow \ldots \varphi_{ab}\,x_n \rightarrow \varphi_{ab}\,y$.

Definition compute (l, y) := member y l

Theorem correctness_list:
$\forall l, y, compute\ (l, y)\ =\ true\ \rightarrow\ \varphi(l, y)$
(par induction sur l)

Goal: $A \rightarrow B \rightarrow A \rightarrow A \rightarrow B \rightarrow A \rightarrow B$
change (φ_{ab} ([a; b; a; a; b; a], b)). apply correctness_list.

(New goal: `compute` ([a; b; a; a; b; a], b) = *true*.)

reflexivity. Qed.



Exemples simplistes (2)

Similaire, en remplaçant n nat par un couple (l, y)

Inductive ab: Set := a : ab | b : ab.

On définit φ_{ab} par $\varphi_{ab}\,\mathsf{a} = A$ et $\varphi_{ab}\,\mathsf{b} = B$,
et φ tel que $\varphi([x_1; \ldots x_n], y) = \varphi_{ab}\, x_1 \to \ldots \varphi_{ab}\, x_n \to \varphi_{ab}\, y$.

Definition compute (l, y) := member y l

Theorem correctness_list:
$\forall l, y, compute\ (l, y)\ =\ true\ \to\ \varphi(l, y)$
(par induction sur l)

Goal: $A \to B \to A \to A \to B \to A \to B$
change (φ_{ab} ([a; b; a; a; b; a], b)). apply correctness_list.

(New goal: `compute` ([a; b; a; a; b; a], b) = *true*.)

reflexivity. Qed.



Exemples simplistes (2)

Similaire, en remplaçant n nat par un couple (l, y)

Inductive ab: Set := a : ab | b : ab.

On définit φ_{ab} par $\varphi_{ab}\,\mathsf{a} = A$ et $\varphi_{ab}\,\mathsf{b} = B$,
et φ tel que $\varphi([x_1; \ldots x_n], y) = \varphi_{ab}\, x_1 \rightarrow \ldots \varphi_{ab}\, x_n \rightarrow \varphi_{ab}\, y$.

Definition compute (l, y) := member y l

Theorem correctness_list:

$$\forall l, y, compute\ (l, y)\ =\ true\ \rightarrow\ \varphi(l, y)$$

(par induction sur l)

Goal: $A \rightarrow B \rightarrow A \rightarrow A \rightarrow B \rightarrow A \rightarrow B$
change (φ_{ab} ([a; b; a; a; b; a], b)). apply correctness_list.

(New goal: `compute` ([a; b; a; a; b; a], b) = *true*.)

reflexivity. Qed.



Plus joli : réflexion minimale

Tout autre domaine cible que les booléens peut convenir

On peut aussi supprimer les booléens
Remplacer *true* = *true* par *True* (et *true* = *false* par *False*)

- plus besoin de rewrite et autre discriminate

Definition *compute* : $code_of_a_prop \to Prop$
Theorem correctness: $\forall x : code_of_a_prop, compute\ x \to \varphi x$

- Calculer avec *True* et *False* au lieu de *true* et *false*
- Piège à éviter : résultats genre $True \wedge (False \vee True) \ldots$

$\longrightarrow$ programmation par continuations

Étape suivante : supprimer False
Voir démo
Le calcul de *compute x* produit soit *True*, soit φx (en cas d'échec).



Plus joli : réflexion minimale

Tout autre domaine cible que les booléens peut convenir

On peut aussi supprimer les booléens
Remplacer $true = true$ par $True$ (et $true = false$ par $False$)

- plus besoin de rewrite et autre discriminate

Definition $compute\ : code_of_a_prop \rightarrow Prop$
Theorem correctness: $\forall x : code_of_a_prop, compute\ x \ \rightarrow\ \varphi x$

- Calculer avec $True$ et $False$ au lieu de $true$ et $false$
- Piège à éviter : résultats genre $True \wedge (False \vee True) \dots$

$\longrightarrow$ programmation par continuations

Étape suivante : supprimer False
Voir démo
Le calcul de $compute\ x$ produit soit $True$, soit φx (en cas d'échec).



Plus joli : réflexion minimale

Tout autre domaine cible que les booléens peut convenir

On peut aussi supprimer les booléens
Remplacer $true = true$ par $True$ (et $true = false$ par $False$)

- plus besoin de rewrite et autre discriminate

Definition $compute$: $code_of_a_prop \rightarrow Prop$
Theorem correctness: $\forall x : code_of_a_prop, compute\ x \ \rightarrow\ \varphi x$

- Calculer avec $True$ et $False$ au lieu de $true$ et $false$
- Piège à éviter : résultats genre $True \wedge (False \vee True) \ldots$

$\longrightarrow$ programmation par continuations

Étape suivante : supprimer False
Voir démo
Le calcul de $compute\ x$ produit soit $True$, soit φx (en cas d'échec).



Plus joli : réflexion minimale

Tout autre domaine cible que les booléens peut convenir

On peut aussi supprimer les booléens
Remplacer $\mathit{true} = \mathit{true}$ par *True* (et $\mathit{true} = \mathit{false}$ par *False*)

- plus besoin de rewrite et autre discriminate

Definition *compute* : $\mathit{code_of_a_prop} \rightarrow \mathit{Prop}$
Theorem correctness: $\forall x : \mathit{code_of_a_prop}, \mathit{compute}\ x \rightarrow \varphi x$

- Calculer avec *True* et *False* au lieu de *true* et *false*
- Piège à éviter : résultats genre $\mathit{True} \wedge (\mathit{False} \vee \mathit{True}) \ldots$

$\longrightarrow$ programmation par continuations

Étape suivante : supprimer False
Voir démo
Le calcul de *compute x* produit soit *True*, soit φx (en cas d'échec).



Plus joli : réflexion minimale

Tout autre domaine cible que les booléens peut convenir

On peut aussi supprimer les booléens

Remplacer $true = true$ par $True$ (et $true = false$ par $False$)

- plus besoin de rewrite et autre discriminate

Definition $compute$: $code_of_a_prop \rightarrow Prop$

Theorem correctness: $\forall x : code_of_a_prop, compute\ x \ \rightarrow\ \varphi x$

- Calculer avec $True$ et $False$ au lieu de $true$ et $false$
- Piège à éviter : résultats genre $True \wedge (False \vee True) \ldots$

$\longrightarrow$ programmation par continuations

Étape suivante : supprimer False

Voir démo

Le calcul de $compute\ x$ produit soit $True$, soit φx (en cas d'échec).



Plus joli : réflexion minimale

Tout autre domaine cible que les booléens peut convenir

On peut aussi supprimer les booléens

Remplacer $true = true$ par $True$ (et $true = false$ par $False$)

- plus besoin de rewrite et autre discriminate

Definition *compute* : $code_of_a_prop \rightarrow Prop$

Theorem correctness: $\forall x : code_of_a_prop, compute\ x \ \rightarrow\ \varphi x$

- Calculer avec *True* et *False* au lieu de *true* et *false*
- Piège à éviter : résultats genre $True \wedge (False \vee True) \ldots$

$\longrightarrow$ programmation par continuations

Étape suivante : supprimer False

Voir démo

Le calcul de *compute* x produit soit *True*, soit φx (en cas d'échec).



Plus joli : réflexion minimale

Tout autre domaine cible que les booléens peut convenir

On peut aussi supprimer les booléens

Remplacer $true = true$ par $True$ (et $true = false$ par $False$)

- plus besoin de rewrite et autre discriminate

Definition $compute\ : code_of_a_prop \rightarrow Prop$

Theorem correctness: $\forall x : code_of_a_prop, compute\ x \ \rightarrow\ \varphi x$

- Calculer avec $True$ et $False$ au lieu de $true$ et $false$
- Piège à éviter : résultats genre $True \wedge (False \vee True) \dots$

$\rightarrow$ programmation par continuations

Étape suivante : supprimer False

Voir démo

Le calcul de $compute\ x$ produit soit $True$, soit φx (en cas d'échec).

